DAPPS WITH SOLIDITY ON ETHEREUM

Aditya Relangi

Wait, what is a Dapp?
Dapps are distributed apps. These are the applications that run on top of blockchains like ethereum and are written in languages like Solidity

Things we won't talk about:

- Hashing
- Mining
- Investing

I just want to make it clear that this is an intermediate level of tech talk.

I'm going to assume that you have some knowledge of how blockchains work, even if you don't know the inner workings of blockchain technologies, you should be able to able to follow this talk

As the title says, here are the things we won't talk about.

Hashing, Mining I expect you to have an understanding of this.

And I'm not going to talk about any alt-coins, ICOs, tokens and other applications

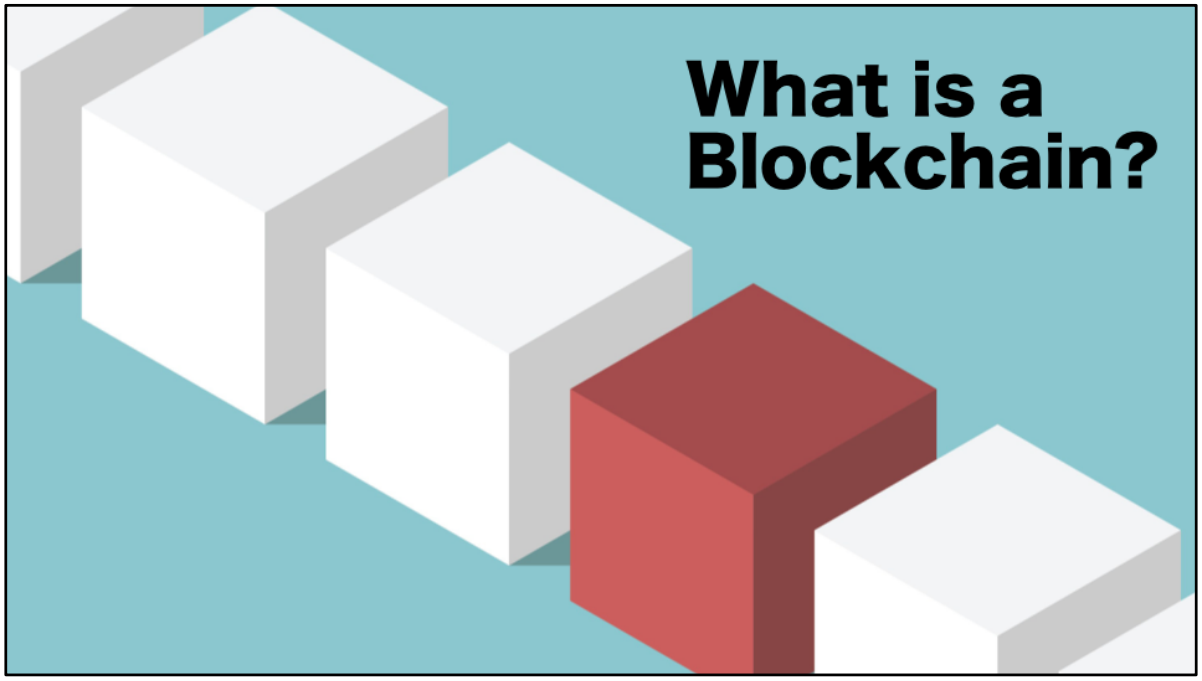
With that, let's look at what we will talk about today.

1. I'll go into what a programmable blockchain
2. What ethereum is and how/why it came about
3. A small intro to Solidity
4. Then we will jump into building a Dapp
5. And finally conclude with how we can go beyond ethereum

6. With that let's get started with the first topic of our talk

Programmable Blockchain

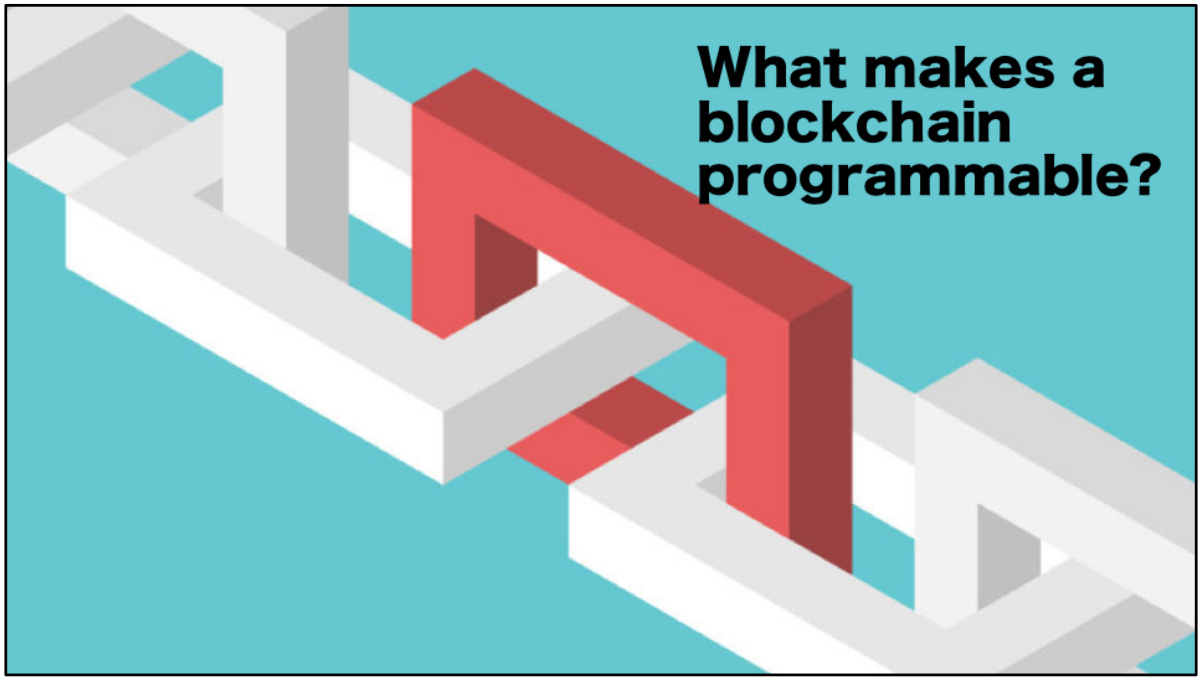
The programmable blockchain, before we get into that, let’s look at what a blockchain means



Before we get into what a programmable blockchain, let's look at what a blockchain is

A blockchain is a bunch of transactions that are validated and grouped together into blocks that are chained sequentially and are secured using cryptography to maintain an immutable history

Apart from the transactions, the blocks typically contain the timestamp info and a hash of the previous block



Quite simply, the ability to execute some programs on the blockchain…

These programs can be simple scripts.

The simplest and the most basic example is **validating transactions**

You can have other programs that'd perform certain actions when certain conditions have been met.

For example, you can have a company that pays it's employees through cryptocurrency, where the money is transferred from the company's payroll account to the employee account on the first of every month

Before we answer the question of whether bitcoin is a programmable blockchain, why should we even talk about bitcoin?

Well, because bitcoin is the first blockchain protocol. So, is it programmable?

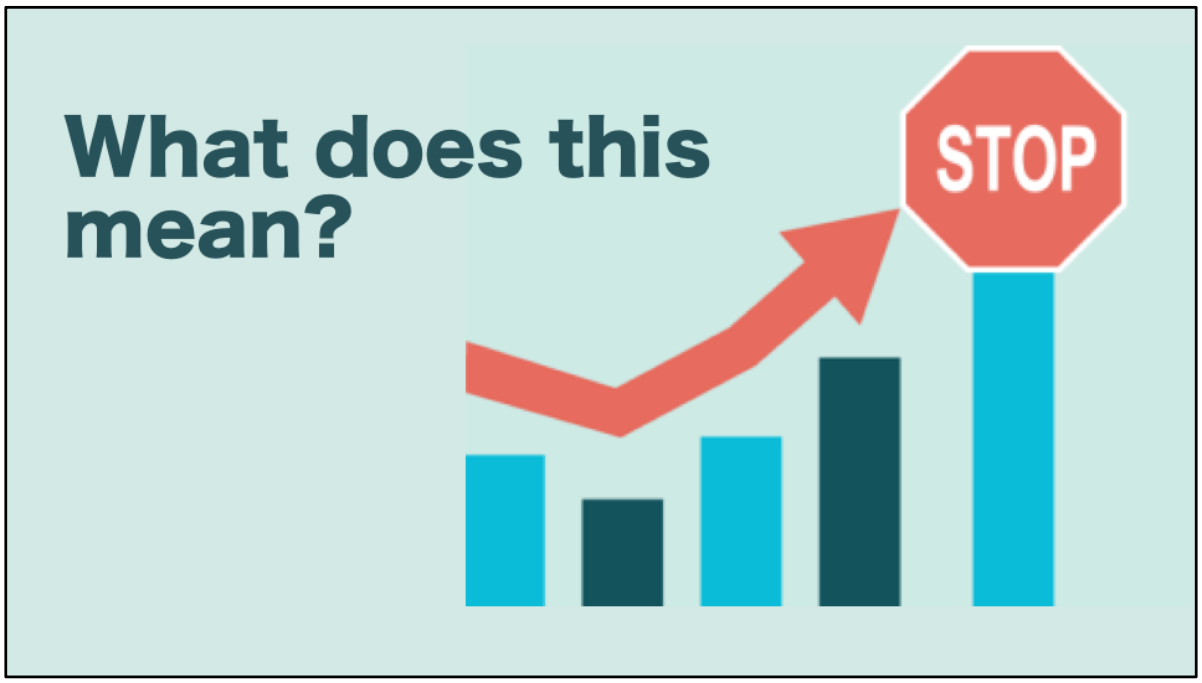
Yes it is.

Bitcoin is programmable through its transactions. Transactions in bitcoin are small programs and most of them do stuff like "relase the bitcoin from this address to address X when provided with the public key for X and a signature proving possession of the private key for this address”

What this does is, it validates the public and private keys and releases the bitcoins to the new address.

- Great right, but there's more to this…
- Even though Bitcoin is programmable, it is not turing complete.
- The bitcoin transactions as I said are simple scripts that get executed.
- These scripts are written using the language, bitcoin script. It is a stack based language that isn't turing complete.
- Specifically there are no loops in bitcoin script....
- Wait, what? Why? Why do we want to handicap a language? Loops are great, we can iterate and do a lot of cool stuff, right?
- But... loops can also be dangerous. The main idea of blockchains is to have a distributed network that validates transactions, create blocks and maintain the entire blockchain.This means that the transaction scripts are running on miners' machines all across the globe
- Because of the distributed nature of the protocol, this poses a unique challenge.
- If the bitcoin script was turing complete you could have scripts that are fairly small but would take a long time to run. Let's say there's an infinte loop in your script, that'd mean the miners' resources are tied up .
- This would render a denial of service attack against everyone on the network when they try to verify the transaction.Instead, if we make our script turing incomplete by avoiding loops, that provides us this functionality without the

complexity.



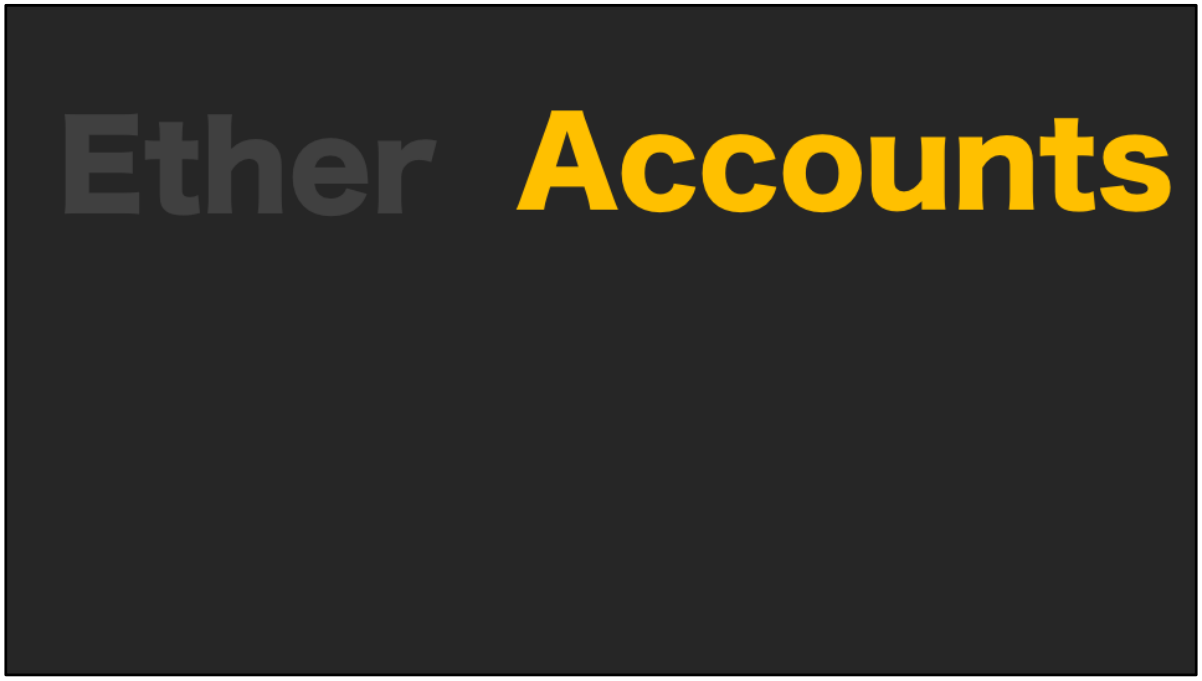
- This means we have this network of distributed computing power that is running all the time, all across the world, that is intentionally limited in its capability.
- Is there a way to remove this limitation and get the benefits of this vast computing power?

Answering that question brings us to the next segment of our talk

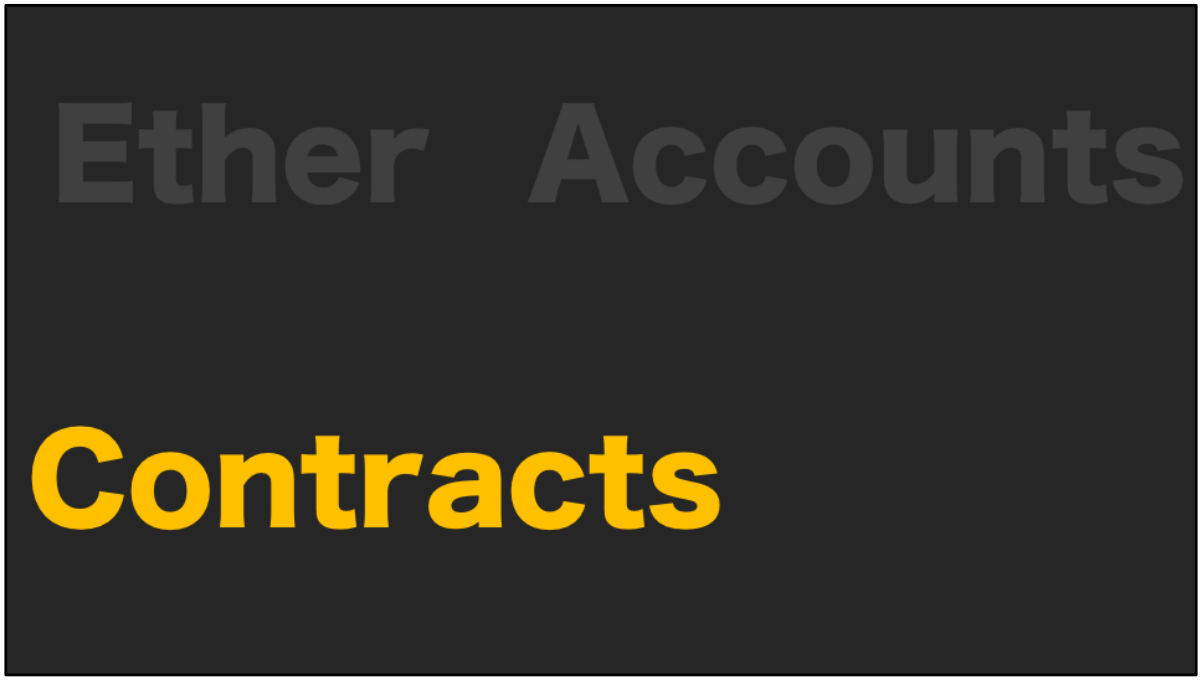
- Ethereum is the ultimate abstraction of the blockchain
- It was proposed by the then 19 year old vitalik buterin in 2013.
- The idea behind ethereum as his quote says is for it to be a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications
- There’s more to Ethereum than what’s included in this quote, let’s go over the important compnonents

Ether

- Ether is the currency of the Ethereum network. Ether has a currency value and is currently at ~900$.
- This is the incentive that Miners get for creating blocks on the Ethereum blockchain



- Accounts represent identities of external agents.
- Accounts use public key cryptography to sign transaction so that the EVM can securely validate the identity of a transaction sender.
- There are two types of accounts
    - Externally Owned accounts – has balance, owned by humans, has no associated code and can send transactions
    - Contract accounts – has balance, has code associated with it, that gets triggered by transactions and can call other contracts



- A contract is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain.
- Contracts are typically written in some high level language such as Solidity and then compiled into bytecode to be uploaded on the blockchain.

- Contracts live on the blockchain in a Ethereum-specific binary format called Ethereum Virtual Machine (EVM) bytecode.
- The Ethereum Virtual Machine or EVM is the runtime environment for smart contracts in Ethereum.
- It is not only sandboxed but actually completely isolated, which means that code running inside the EVM has no access to network, filesystem or other processes.
- EVM is kinda like JVM, many of you must be familiar with it.
- All the JVM does is, it executed this bytecode that was compiled from a high level language, like solidity

- Contracts live on the blockchain in a Ethereum-specific binary format called Ethereum Virtual Machine (EVM) bytecode.
- EVM is kinda like JVM, many of you must be familiar with it.
- All the JVM does is, it executed this bytecode that was compiled from a high level language, like solidity

- Along with the concepts that we just spoke about, let’s take a look again at Buterin’s quote again
- The thing I’d like you to pay attention to is that this supports Turing-complete languages
- Turing-complete languages like Solidity, that support loops
- But we learned from earlier that there’s a problem when there are loops which bitcoin altogether avoids, that problem is..

What

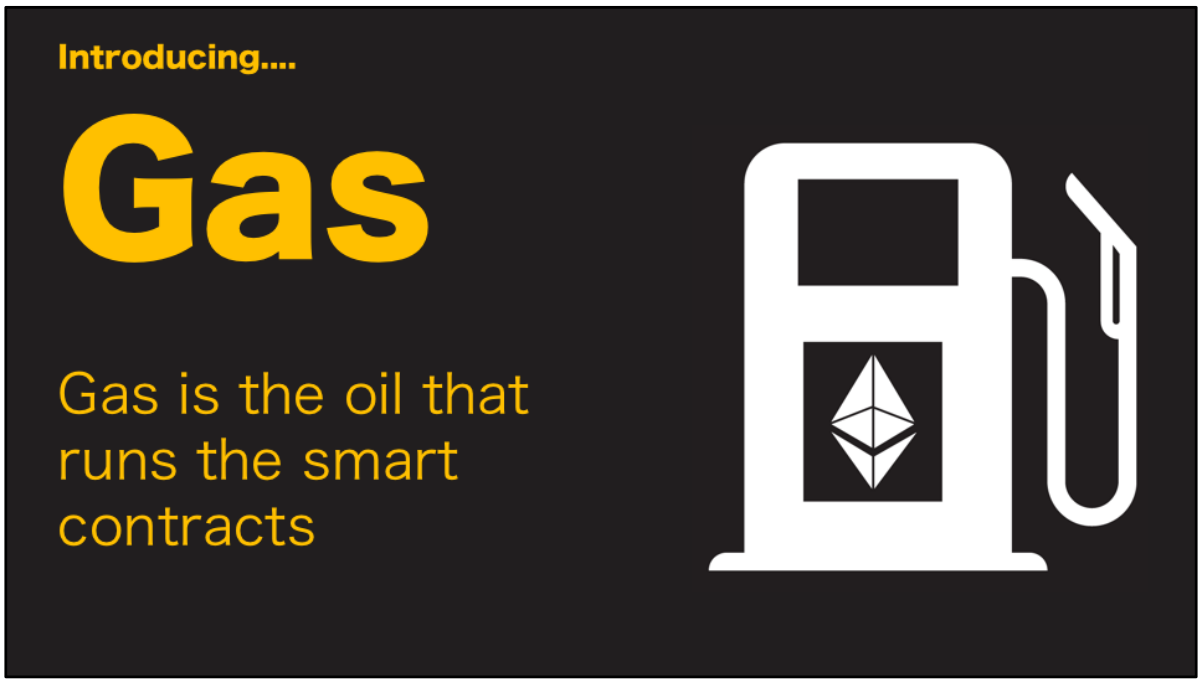
What about

What about the

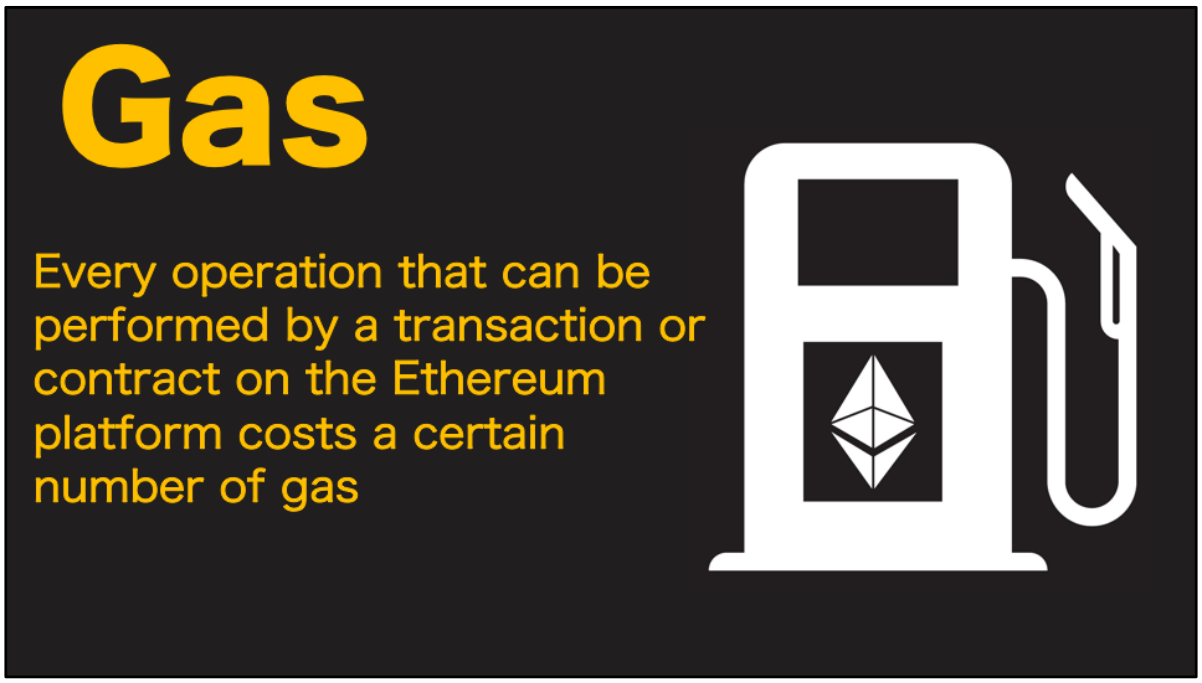
What about the infinite

What about the infinite loops?

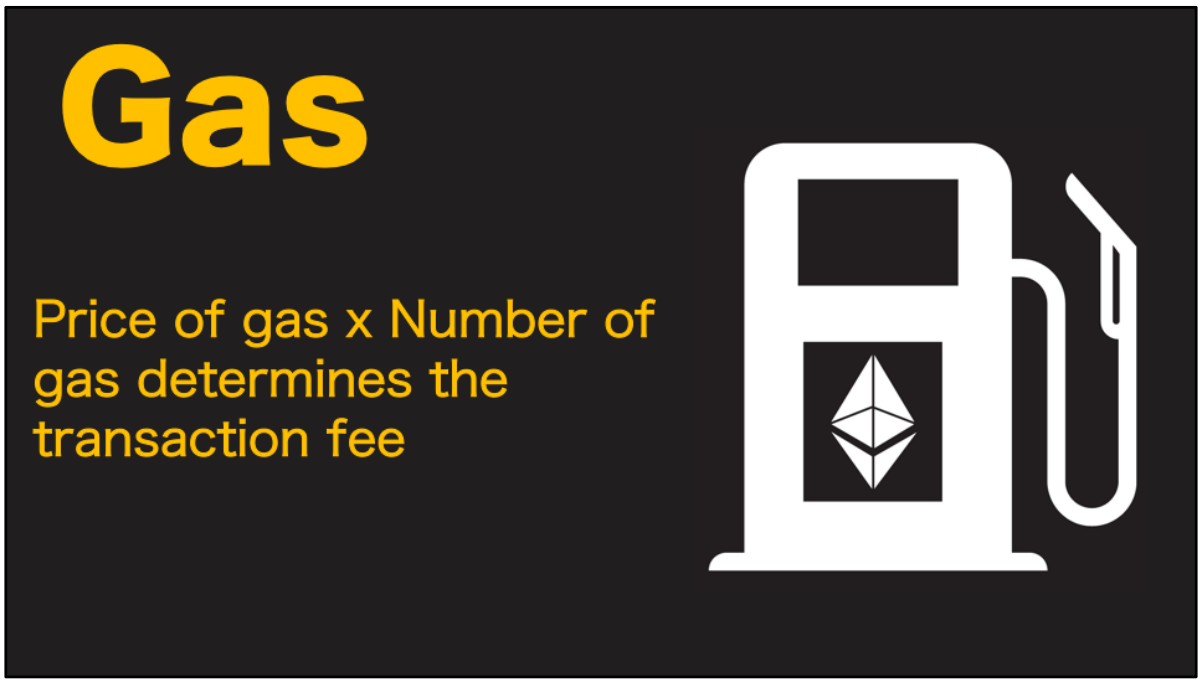
- Well, that is where the ingenuity of Ethereum shines through
- You see, the problem with infinite loops or computations that take really long for miners is that their computation power is wasted and they do not get any reward for running these programs.
- Well, what if the miner gets paid for running your transactions and they charge you for running your scripts?
- That's sounds reasonable right?
- Cool, let's see how ethereum does it



- Gas is an interesting concept that ensures that the miners are compensated for providing the computational resources while the users stay responsible for their programs



- Gas is an interesting concept that ensures that the miners are compensated for providing the computational resources while the users stay responsible for their programs
- Ethereum allows arbitrarily complex computer code to be run, a short length of code can actually result in a lot of computational work being done.
- So it's important to measure the work done directly instead of just choosing a fee based on the length of a transaction or contract.
- Although gas is a unit that things can be measured in, there isn't any actual token for gas.
- That is, you can't own 1000 gas. Instead, gas exists only inside of the Ethereum virtual machine as a count of how much work is being performed.
- When it comes to actually paying for the gas, the transaction fee is charged as a certain number of ether, the built-in token on the Ethereum network and the token with which miners are rewarded for producing blocks.



- Gas is the way that fees are calculated
- The fees are still paid in ether, though, which is different from gas
- The gas cost is the amount of work that goes into something, like the number of hours of labour, whereas the gas price is like the hourly wage you pay for the work to be done.
- The combination of the two determines your total transaction fee.
- If your gas price is too low, no one will process your transaction
- If your gas price is fine but the gas cost of your transaction runs "over budget" the transaction fails but still goes into the blockchain, and you don't get the money back for the work that the labourers did.
- This makes sure that nothing runs forever, and that people will be careful about the code that they run.
- It keeps both miners and users safe from bad code!

- https://ethereum.stackexchange.com/questions/3/what-is-meant-by-the-term-gas

Answering that question brings us to the next segment of our talk

Solidity Features

- High-level language
- Statically typed
- Inheritance
- User-defined types
- Libraries

- Solidity is a contract-oriented, high-level language for implementing smart contracts.
- It was influenced by C++, Python and JavaScript and is designed to target the Ethereum Virtual Machine (EVM).
- Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.
- Like objects in OOP, each contract contains state variables, functions, and common data types.
- Contract-specific features include modifier clauses, event notifiers for listeners, and custom global variables.

```
// Declare the compiler version
pragma solidity ^0.4.19;

/* 'contract' has similarities to 'class' in other languages */
contract SimpleStorage {
    uint public storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public constant returns (uint retVal) {
        return storedData;
    }
}
```

- All solidity source code should start with a "version pragma" — a declaration of the version of the Solidity compiler this code should use.
- This is to prevent issues with future compiler versions potentially introducing changes that would break your code.

- 'contract' has similarities to 'class' in other languages (class variables,inheritance, etc.)

- Variable declared outside functions are global to the contract

- 'public' makes externally readable (not writeable) by users or contracts and is the default mode. **External, Internal, Inherited, Private**

- ‘constant’ here specifies that the function doesn’t alter the state and so there’s no need to spend gas and doesn’t need gas to be spent

Solidity Features

- Types
- Control Structures
- Special Functions and Variables
- Inheritance
- Abstract Contracts
- Fallback Functions/Function Modifiers
- Events

- **Bool, integer, string, byte** types both signed and unsigned with various bit widths and addresses of 160 bits width
- Fixed and Dynamically sized **arrays** are available, so are **user defined structs**.
- Ether and Time units
    - **wei, finney, szabo or ether** are the currency types. Wei is the minimum unit. 10^18 wei = ether
    - **seconds, minutes, hours, days, weeks and years** are the time units. It works a little naviely, by converting 1 year to 365 days.

Solidity Features

- Types
- Control Structures
- Special Functions and Variables
- Inheritance
- Abstract Contracts
- Fallback Functions/Function Modifiers
- Events

- It also supports a lot of control structures like
  if, else, while, for, break, continue, return. No **switch** or **go to**

Solidity Features

- Types
- Control Structures
- Special Functions and Variables
- Inheritance
- Abstract Contracts
- Fallback Functions/Function Modifiers
- Events

- **block info, msg info, tx info** can be gotten through variables. Cryptographic function for hashing like **sha3, sha256, etc** are also available.
- Things like the block difficulty, block gas limit, blockhash
- Address of the msg sender, etc

Solidity Features

- Types
- Control Structures
- Special Functions and Variables
- Inheritance
- Abstract Contracts
- Fallback Functions/Function Modifiers
- Events

- Solidity supports multiple inheritance by copying code including polymorphism

Solidity Features

- Types
- Control Structures
- Special Functions and Variables
- Inheritance
- Abstract Contracts
- Fallback Functions/Function Modifiers
- Events

- Contract functions can lack an implementation
- Such contracts cannot be compiled (even if they contain implemented functions alongside non-implemented functions), but they can be used as base contracts
- If a contract inherits from an abstract contract and does not implement all non-implemented functions by overriding, it will itself be abstract.

Solidity Features

- Types
- Control Structures
- Special Functions and Variables
- Inheritance
- Abstract Contracts
- Fallback Functions/Function Modifiers
- Events

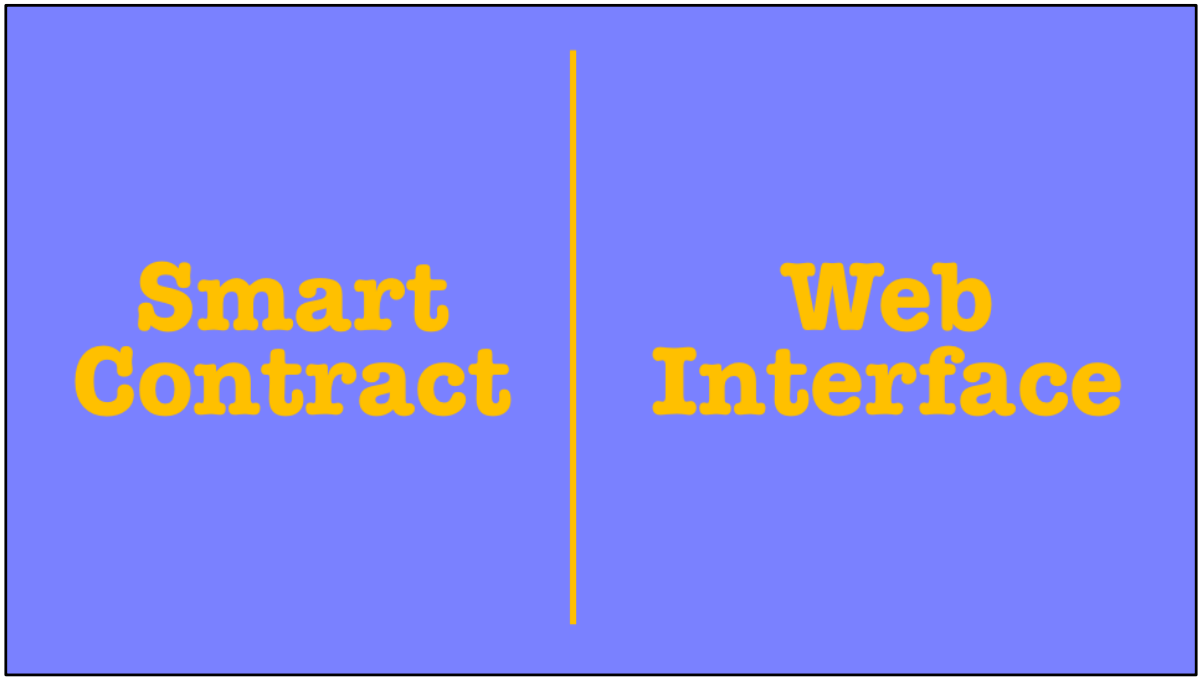
- **Fallback Function** - A contract can have exactly one unnamed function. This function cannot have arguments and is executed on a call to the contract if none of the other functions matches the given function identifier
- **Function modifiers** - Modifiers can be used to easily change the behaviour of functions, for example to automatically check a condition prior to executing the function. They are inheritable properties of contracts and may be overridden by derived contracts.

Solidity Features

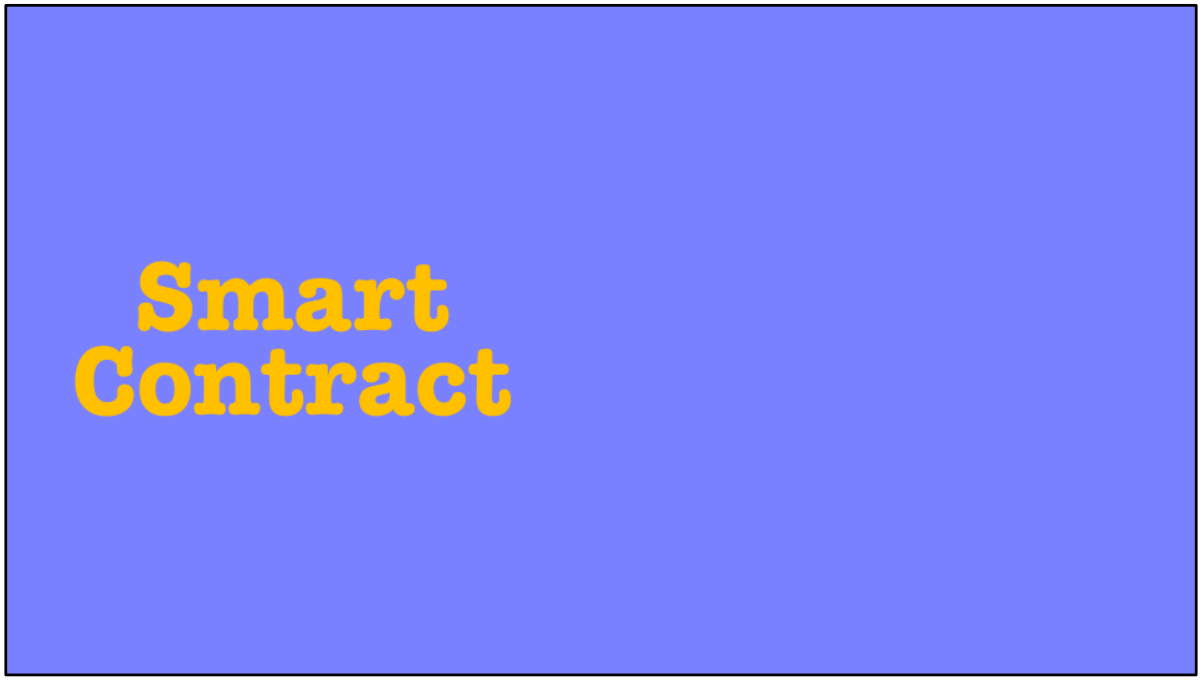
- Types
- Control Structures
- Special Functions and Variables
- Inheritance
- Abstract Contracts
- Fallback Functions/Function Modifiers
- Events

- **Events** are a way for your contract to communicate that something happened on the blockchain to your app front-end, which can be 'listening' for certain events and take action when they happen

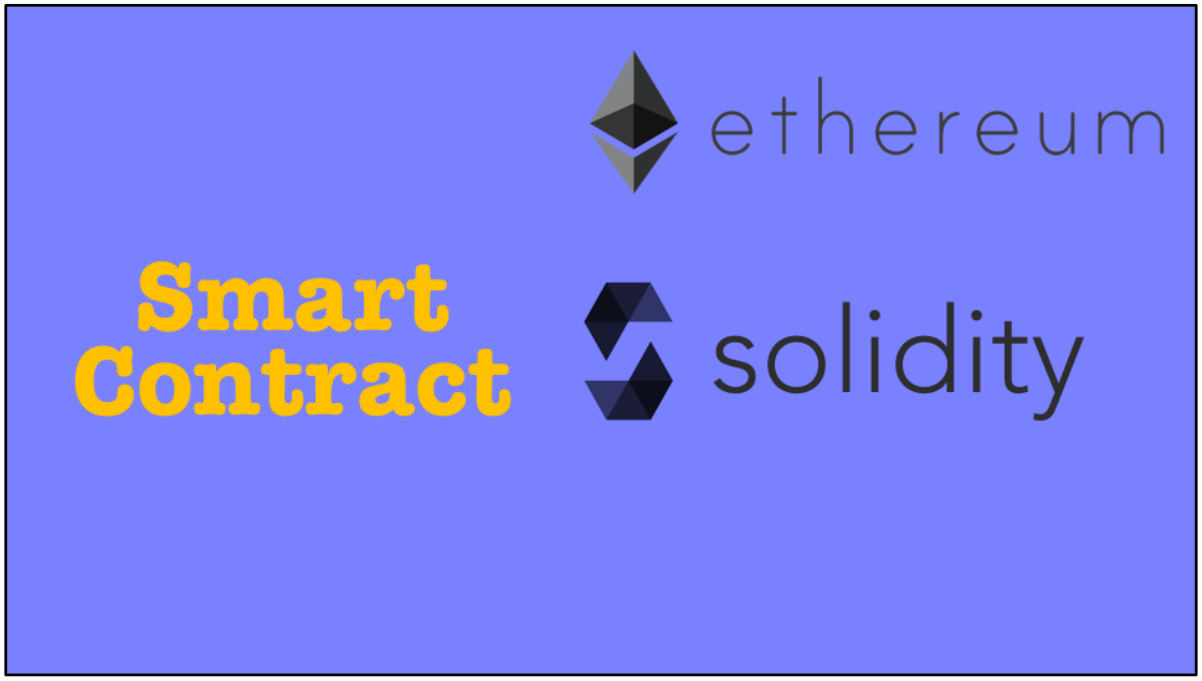

Decentralized app
~~Dapp~~



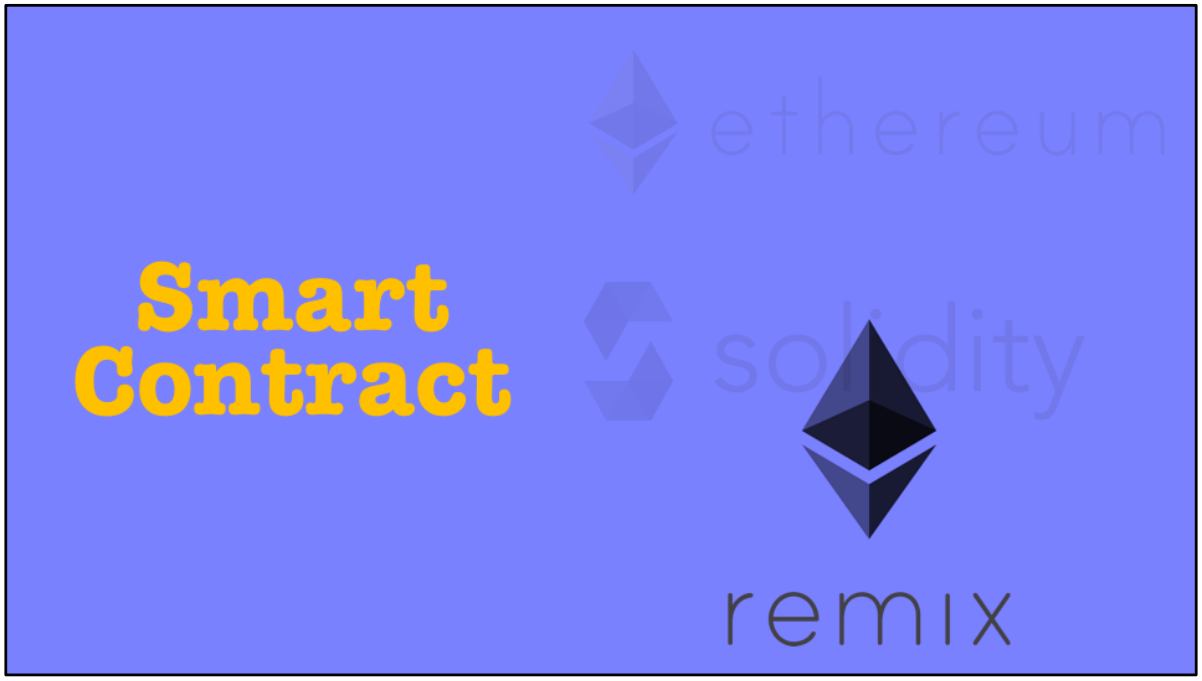
- DApp is an abbreviated form for *decentralized application*.
- A DApp has its backend code running on a decentralized peer-to-peer network. This is different from an app where the backend code is running on centralized servers.
- A DApp can have frontend code and user interfaces written in any language (just like an app) that can make calls to its backend. Furthermore, its frontend can be hosted on decentralized storage such as Swarm or IPFS, but in our case



- Let’s take a look at what we need for the smart contract
- We need the infrastructure to run our smart contract, we know it’ll run on the Ethereum blockchain
- We know the language we are going to write our contract in
- We need an IDE to write our code in. Solidity is supported by a lot of editors like vs code, eclipse, or the greatest editor ever **vim**



- Let's take a look at what we need for the smart contract
- We need the infrastructure to run our smart contract, we know it'll run on the Ethereum blockchain
- We know the language we are going to write our contract in
- We need an IDE to write our code in. Solidity is supported by a lot of editors like vs code, eclipse, or the greatest editor ever **vim**



- Let's take a look at what we need for the smart contract
- We need the infrastructure to run our smart contract, we know it'll run on the Ethereum blockchain
- We know the language we are going to write our contract in
- We need an IDE to write our code in. Solidity is supported by a lot of editors like vs code, eclipse, or the greatest editor ever **vim**
- but for our purposes, we are going to go with **remix**

- The other thing we need is a browser that will interact with the ethereum blockchain, there are some browsers like **brave** dedicated to this, but we will be using the **metamask** chrome plugin which will communicate with the blockchain for us

- web3 which is a javascript library that lets you perform many Ethereum functions

- I'm using Infura as my fallback "provider" and as far as I understand, they are providing the infrastructure required to connect with the blockchain, read/modify it, etc. The fallback provider allows people who don't have tools like MetaMask to at least read the contract state.

- *abiArray* is the Contract Application Binary Interface (ABI) which is basically a description of the function methods and variables. If you go to the code tab on Etherscan, you can find the ABI

- There are probably a few things that are confusing here, such as "wei" and BigNumber.

- As I mentioned earlier, 10^{18} wei = 1 ETH. Smart contracts always transact with wei, and wei is the smallest possible unit of ETH.

- Since these numbers are huge, we need the BigNumber library to handle them properly.

- First, I converted the user provided ETH number into wei. Then, I created

a transactionObject which contains the value parameter. Finally, I called addUser with the supplied string and transaction info.

```
if (typeof web3 !== 'undefined') {
  web3 = new Web3(web3.currentProvider);
} else {
  // set the fallback provider you want from Web3.providers
  web3 = new Web3(new Web3.providers.HttpProvider("https://mainnet.infura.io/2tXmBfvMC1sfg10iQAm4 "));
}

//Connect to the contract
var contract = web3.eth.contract(abiArray).at("0xF4fc13034c1347E3ab1EE1a01c58796bdB20c639");

//Read the contract state
contract.getUser(0,function(error, result) {
          console.log(result);
});


// Call the smart contract to get on the leaderboard
function getonLeaderboard() {
  var ethValueString = $('#ethInput').val();
  var userName = $('#nameInput').val();

  var ethNumber = new BigNumber(ethValueString);
  var weiValue = web3.toWei(ethNumber, 'ether');

  var transactionObject = {
    "value": weiValue
  }

  contract.addScore(userName, transactionObject, function(error, result) {
    console.log(result);
  });
}
```

```
[{
    "constant": false,
    "inputs": [{
        "name": "name",
        "type": "string"
    }],
    "name": "addScore",
    "outputs": [{
        "name": "",
        "type": "bool"
    }],
    "payable": true,
    "stateMutability": "payable",
    "type": "function"
}, {
    "constant": true,
    "inputs": [{
        "name": "index",
        "type": "uint256"
    }],
    "name": "getUser",
    "outputs": [{
        "name": "",
        "type": "address"
    }, {
        "name": "",
        "type": "uint256"
    }, {
        "name": "",
        "type": "string"
    }],
    "payable": false,
    "stateMutability": "view",
    "type": "function"
}]
```

Answering that question brings us to the next segment of our talk

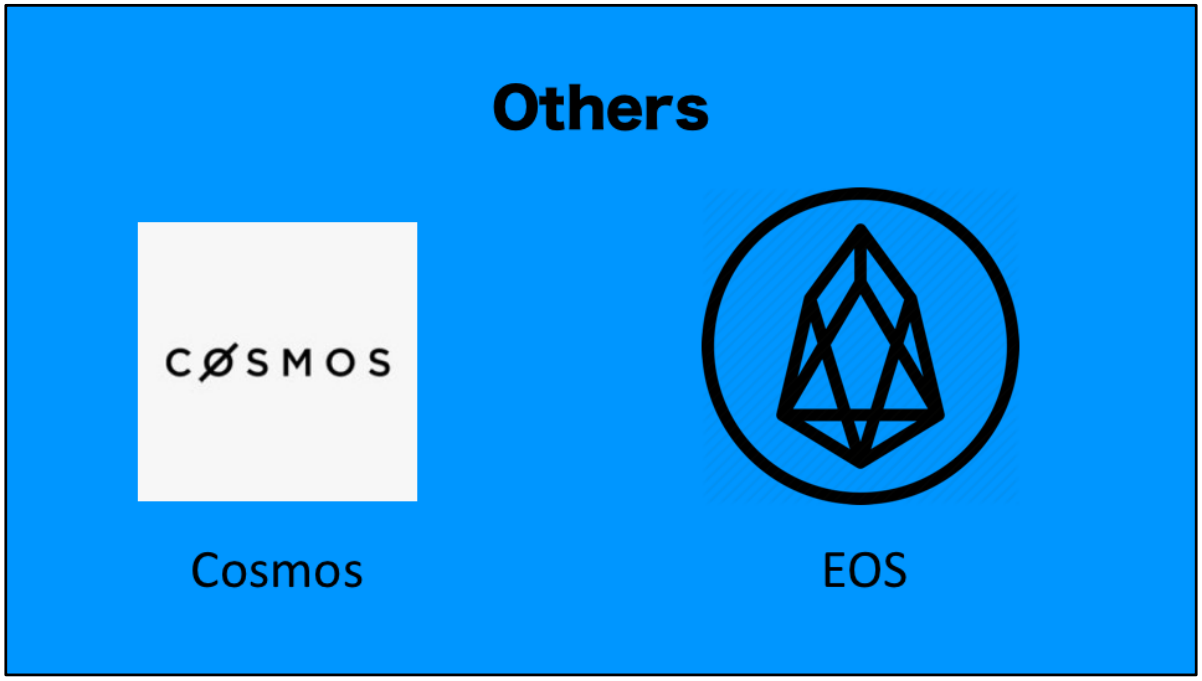
What about the problems or pitfalls

Immutability

Correctness

- There are a couple of issues with smart contracts and they are immutability and correctness
- The contract we published has a bug
- There is no way to change this contract now, we will have to publish a new contract to correct this
- Because of this correctness is a really important thing

Alright we can now go on to the next segment



- There are alternatives to Ethereum to build applications
- Of course I'm not mentioning stuff like hyperledger, fabric, corda, etc., but you get the idea

Languages/Frameworks

- Serpent, LLL
- Truffle, Embark
- Dapp
- Dappsys
- IPFS

- Dapp is a simple command line tool for smart contract development
- simple, flexible building-blocks for smart-contract systems
- Truffle is a development framework
- Ember Framework for serverless Decentralized Applications using Ethereum, IPFS and other platforms

Resources

- Ethereum
- Solidity
- Dapp
- Code repo
- Etherscan
- Solidity book
- Gas

Alright we can now go on to the next segment